

# A Prototype Grid-site Monitoring System

Version 1, January 2002.

Mark Baker, Hong Ong, Garry Smith,  
Distributed Systems Group,  
University of Portsmouth,  
Portsmouth, UK

1	Abstract .....	4
2	Introduction .....	4
3	Architecture .....	5
4	Prototype High-level Requirements .....	7
5	Current Implementation .....	8
5.1	Client-side Functionality .....	8
5.1.1	Main Applet Window .....	8
5.1.2	Grid Site Map .....	8
5.1.3	Site Information .....	10
6	Libraries and APIs .....	13
7	Data Structures .....	14
7.1	Meta data .....	14
7.2	Data Structure Architecture .....	15
8	Client-Server Communications .....	16
8.1	Timestamps: Client-Server Data Control .....	17
9	Servlet MDS Polling Approach .....	18
9.1	MDS Polling Mechanism .....	18
9.2	Polling Periods .....	18
10	Local Database .....	19
10.1	Database Tables .....	19
11	Future Work .....	21
11.1	Security .....	21
11.2	Plotting Map Positions .....	21
11.3	The User Interface .....	21
11.4	Further Multi-Threading .....	21
12	Summary .....	22
13	References .....	22
14	Appendix A: Local Database Configuration .....	23
14.1	SQL Syntax .....	23
14.2	Additional Table Definitions .....	24
14.2.1	Dynamic Tables .....	25

Figure 1: Grid Site Monitoring Architecture .....	5
Figure 2: Grid Site Location and Status Map .....	9
Figure 3: Grid Site Location Map – Summary Information Displayed by Mouse-over Events.....	9
Figure 4: Grid Site – Display Registered Site Data .....	10
Figure 5: Polling Information From MDS Nodes .....	11
Figure 6: Query MDS for Compute Node Information.....	12
Figure 7: Runtime Events Displayed.....	13
Figure 8: The Client Data Retrieval Mechanism .....	16
Table 1: Grid Monitor Prototype – Requirements .....	7
Table 2: Grid Site Map – Status Icons Key .....	8
Table 3: Libraries and APIs .....	14
Table 4: Examples of Meta Data Grouping .....	15
Table 5: AdminTable – Database Table .....	20
Table 6: AdminContactTable – Database Table .....	20
Table 7: MdsServerTable – Database Table.....	21

## Abstract

The UK Grid Support Centre, established by the DTI Core Programme for e-Science [e-science02], supports the deployment, operation and maintenance of Grid middleware and distributed resource management for the UK Grid test beds. [uk-grid-support02]. The Support Centre operates a registration service and project database for participating UK Grid test bed sites.

This report presents a prototype Grid Monitor that allows the monitoring of Grid information servers at sites registered with the UK Grid Support Centre. The prototype presents metadata from individual Grid sites in a coherent, easy to use format.

## 1 Introduction

The software prototype presented is a three-tier Web utility designed to help Grid administrators and users keep track of the availability and loading of Grid resources within a Grid-enable site. The prototype's architecture consists of an applet, a Servlet engine [tomcat02] and a relational database [mysql02]. The Globus Java Commodity Grid Kit (CoG kit) APIs [cog02] are used to communicate with Grid Information Servers [globus-mds02].

The Central Laboratory for the Research Councils (CLRC) at Daresbury Laboratory, as part of the UK e-Science programme (UK Grid Support Centre), sponsored the work on this prototype.

### *Functionality at a Glance*

- Display Grid information server status for all sites: Colour coding denotes the availability of metadata servers (Globus MDS (Metacomputing Directory Service) [globus-mds02]).
- Site-specific information (site and contact details, MDS server details, server and compute resource availability).

### *Future (Functional) Enhancements*

- Plots of historical CPU loads for each site's compute resources.
- Histograms of general site availability (i.e. MDS)
- Network load between Grid Sites

## 2 Architecture

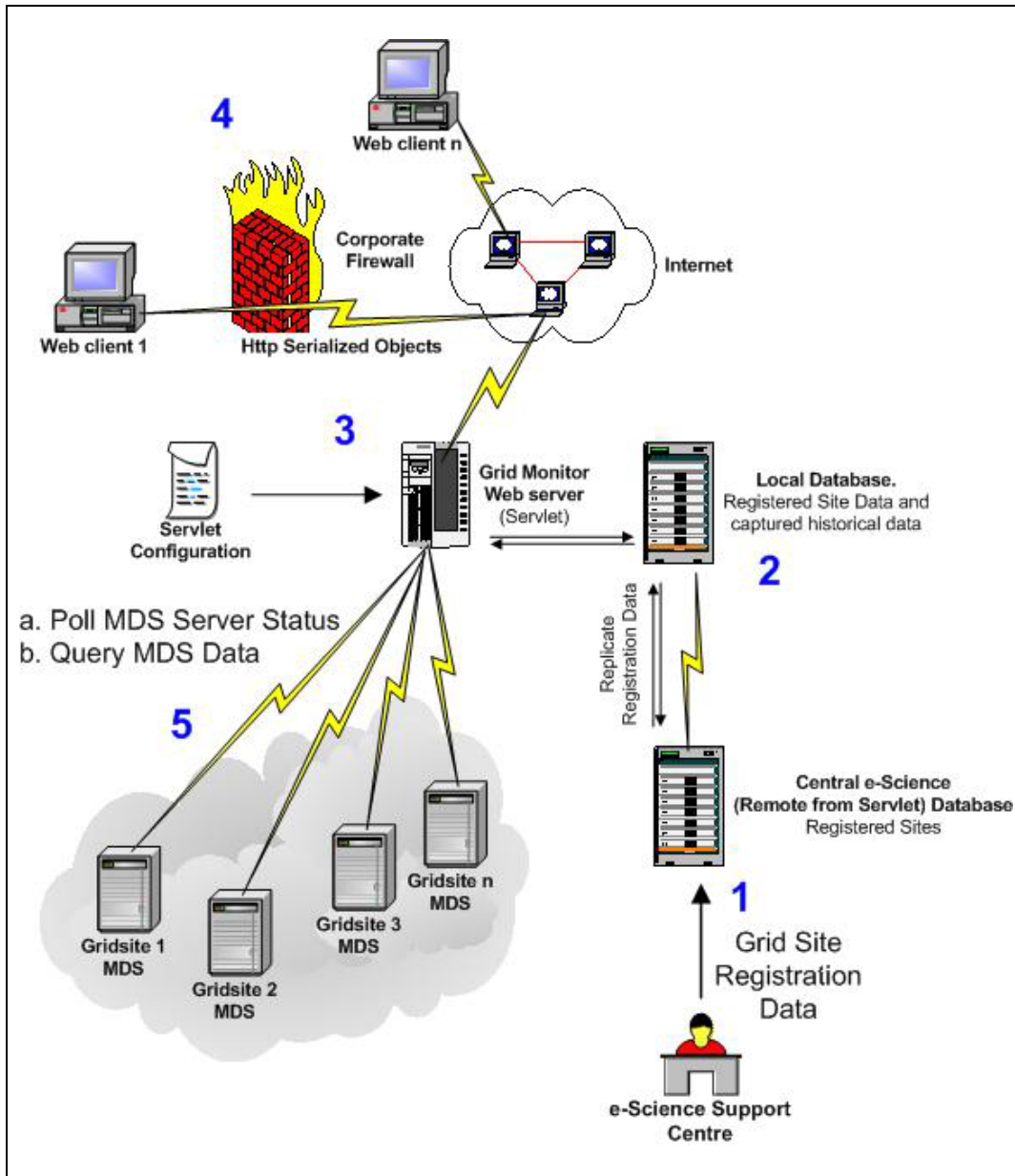


Figure 1: Grid Site Monitoring Architecture

The Grid site Monitoring architecture is a three-tier architecture and is shown in Figure 1. The tiers comprise of the following elements:

- Web clients (user interface),
- Web server/servlet (application logic),
- Relational database (repository of Grid site registration and historical data).

The Grid monitoring architecture is referenced using the integers in Figure 1:

1. When an organisation registers with the e-Science Support Centre, site metadata is required. For example, the site name, address, contact details, support staff and name of Grid Information Servers (GIS). This information is stored (and updated where necessary) within a central database maintained by the e-Science Support team. This database acts as a data source to the Grid site monitoring software and provides the information required to monitor each site (i.e. GIS hostname and port numbers).
2. The Grid monitoring software utilises a local database. This database performs two tasks:
  - i. Acts as a mirror (cache) for the central e-Science database, providing Grid site information directly to the servlet.
  - ii. Stores historical data collected during GIS polling.

This distributed database approach is intended to

- iii. Reduce the effects of network (Internet) latency by moving the required data close to the servlet.
- iv. Introduce data redundancy (and therefore availability) through replication. It is anticipated that registered site data will remain reasonably static. Any changes that do occur (i.e. adding a new site, or updating an existing site's data) will be propagated through to the Grid Monitor after a specified update period (or as the result of a notification message).

Data consistency (between the databases) is not viewed as a serious problem; the Grid Monitor requires registration data only. Because updated data is passed from the central database, the primary issue is related to stale data. Data staleness would be controlled by the database synchronisation approach used (i.e. periodic client polls versus a server, push mechanism). The Grid Monitor Prototype does not directly implement database-mirroring techniques. Mirroring is considered to be a generic problem and therefore not addressed in this report.

3. The servlet reads site registration data from the local database. This data is used to periodically connect to each MDS to determine if the MDS service is available. Failures are logged to the local database. Clients connect to the servlet and request monitoring information. Static data (i.e. a Grid site address) is returned to the client. Dynamic data, (i.e. a MDS query to a specific server) is executed on behalf of the client. Results are then passed back from the servlet to the client. During normal operation the servlet records the availability of MDS servers to the database.
4. The user interface employs Java applets. Under normal security constraints, applets may only communicate with the server from which they originated. Therefore the servlet is used as a client proxy for all interaction with Grid sites (indeed any other remote machine). In addition, all communication with the servlet is initiated by the applet. HTTP servlets are unable to directly connect to an applet (applet security restrictions apply), they may only respond to client requests. Therefore, the Grid monitoring software utilises a client-pull communication model. In order to reduce the issues of operating through firewalls, serialised HTTP Java objects are used to communicate with the servlet. If a firewall is configured to allow HTTP traffic, then the Grid Monitor will operate correctly. Due to the applet client-pull approach to

communication, all connection requests originate from inside the firewall. It is not necessary to configure the firewall to accept external connection requests from the servlet as none are made.

5. Globus MDS servers are periodically polled. Although such an approach does not reflect the availability of Grid compute resources at a given site, it does highlight the pattern of availability of the MDS servers. Without prior knowledge of compute resources, MDS servers provide the mechanism to locate and utilise these resources. A failed MDS may equate to the unavailability of an entire site's compute resources.

### 3 Prototype High-level Requirements

#	Requirement	Detail
1	Monitor each of the registered Grid site MDSs (periodically) to determine if they are alive.	Capture this information to a database of historical information
2	Graphical Map displaying site locations and MDS status.	For each grid site on the map, colour the sites icon appropriately to determine current status, i.e. up/down/recently down/not responded since initialisation.
3	A user will be able to click a grid site icon on the map to retrieve site specific information:	Administrator contact details
		MDS host data and availability (polled)
		List of machines queried from the site's MDS (including host and performance data, i.e. OS type, CPU load, etc)
4	Each compute resource within a MDS may be selected on an individual basis for the purpose of gathering detailed data over time.	<b>Periodic probes: Current/recent status: availability and load</b>
		<b>Background polling for a specified period: data to local database for later analysis</b>
		<b>Real-time (user-defined) probing: on screen graphs of current status and performance data</b>
5	Applet based SQL interface to the local servlet database	<b>Allow users to create custom data requests and view the results within the client interface.</b>
6	A Grid site may register a number of MDS servers	These will be treated as primary, secondary servers, etc. If a primary server fails the Grid Monitor will utilise the secondary server and record the failure. Historical MDS availability will be presented graphically upon request.

Table 1: Grid Monitor Prototype – Requirements

Not all of the requirements shown in Table 1 have been implemented in the current version of the prototype. The bold text indicates areas requiring further work. In requirement 4, currently data gathering is not automated. MDS data is presented to

the user only on direct demand (the result of a button press). In requirement 5 the user interface requires constructing. The communications and database access code is already in place.

## 4 Current Implementation

### 4.1 Client-side Functionality

This section presents Grid Monitor functionality from a user's perspective. Examples of the current user interface are given as well as details of specific changes planned for future versions.

#### 4.1.1 Main Applet Window

The main applet window displays welcome text and provides access to the client menu. Currently the menu contains a link to help pages. In subsequent versions of the software the menu will provide access to client configuration mechanisms. For example, to modify the frequency in which the applet contacts the servlet to retrieve results from MDS polling.

#### 4.1.2 Grid Site Map

When the applet is started a Grid site map is spawned in a separate window. The map displays the geographic location for each Grid site. In addition, the status of each site's MDS servers is shown by the colour of the icon representing that site on the map. See Table 2 for image map colours and Figure 2 for the image map. The example in Figure 2 contained a single registered Grid site when the screenshot was captured. The client updates the map automatically, as new sites are added to the registration database.

Nature of problem	Number of site MDS servers	Image map icon status
MDS failure since last poll	Single	Red
	Multiple	Yellow
Unable to contact MDS since servlet initialisation	Single	Black
	Multiple	

Table 2: Grid Site Map – Status Icons Key

Failure to access a MDS results in the following actions:

1. An error is logged (in the historical database) and displayed to the client.
2. If a site maintains a single MDS, that site's image map icon is set to red.
3. If a site maintains multiple MDSs and at least one other MDS is responding correctly, then the site's image map icon is set to yellow.
4. If the servlet has not been able to access a site's MDS at all (since servlet initialisation), then the site's image map icon is set to black.

The Grid Monitor is not constrained to a single map, different maps can be used, but the location of a site within each new map must be calculated in advance (and stored with the site registration data). Although Global Positioning System (GPS)

coordinates are stored by the registration database, these are not yet used to locate Grid sites automatically within a map. With some minor modification, the Grid Monitor could be used with a hierarchy of maps, for example displaying Grid sites from a global perspective right down to country or even county level. In this example, the graphic of the British Isles was taken from a Grid monitoring project at the University of Glasgow [ppepc01].



Figure 2: Grid Site Location and Status Map



Figure 3: Grid Site Location Map – Summary Information Displayed by Mouse-over Events

The text at the bottom of the map (Figure 3) changes to reveal the name of the site when the mouse pointer passes over a site icon.

### 4.1.3 Site Information

Selecting a Grid site causes detailed information for that site to be displayed. To reduce network traffic, the user must elect to view this data by pressing the button shown in Figure 4. When the panel first loads it is populated with summary information only. The information displayed here is an example of a subset of the data supplied when the Grid site was registered with the central support centre database.

Registration Information includes (not necessarily shown):

- Site Name,
- Postcode and or GPS coordinates,
- Administrators names,
- Contact telephone number,
- GIIS or GIISes IP addresses and ports (GIIS port does not appear standardised),
- GRIS or GRISs IP address and port (to be on the safe side),
- URLs and descriptions of related projects and user documentation.

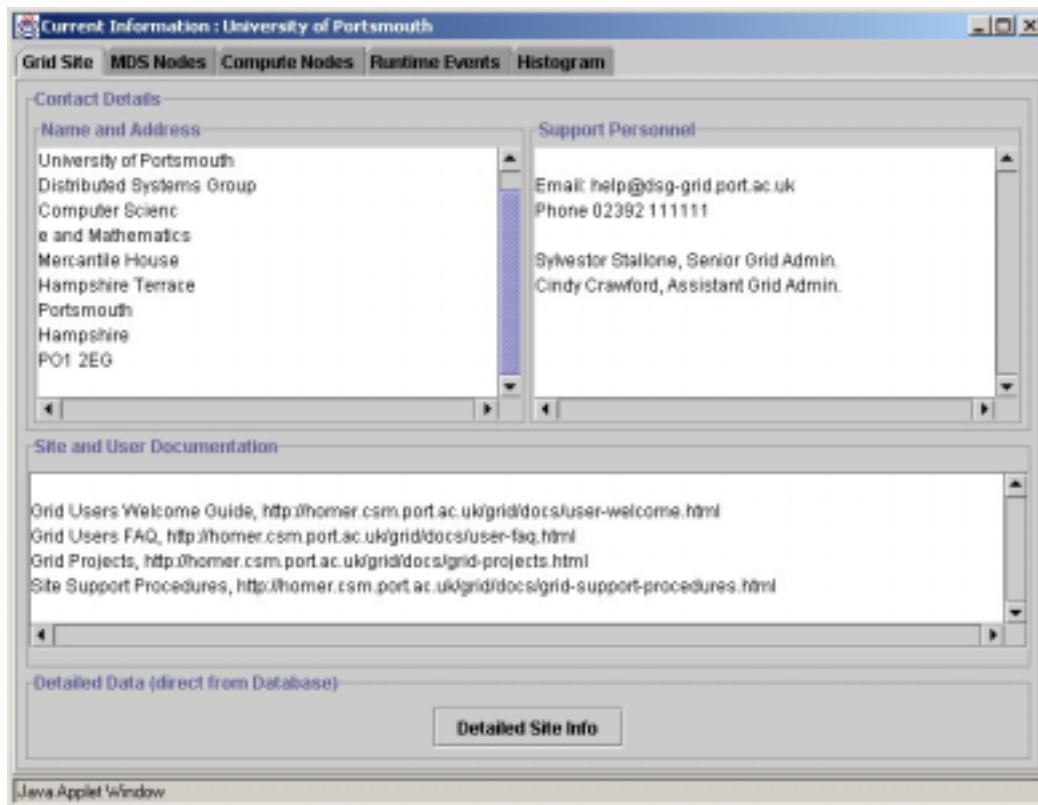


Figure 4: Grid Site – Display Registered Site Data

In a later version of the Grid monitoring software, the display in Figure 4 can be enhanced to provide hyperlink access to documents in the site and user documentation list.

Figure 5 shows MDS node information for the selected Grid site. This information is retrieved directly from cached data at the client. The polling frequency of an MDS is determined by the servlet configuration. Although clients can potentially adjust the period at which they individually connect to the servlet for data updates, the frequency by which the servlet polls an MDS site is currently the same for all Grid sites. Figure 5 indicates that both MDSs for the selected site were alive when last polled by the servlet.

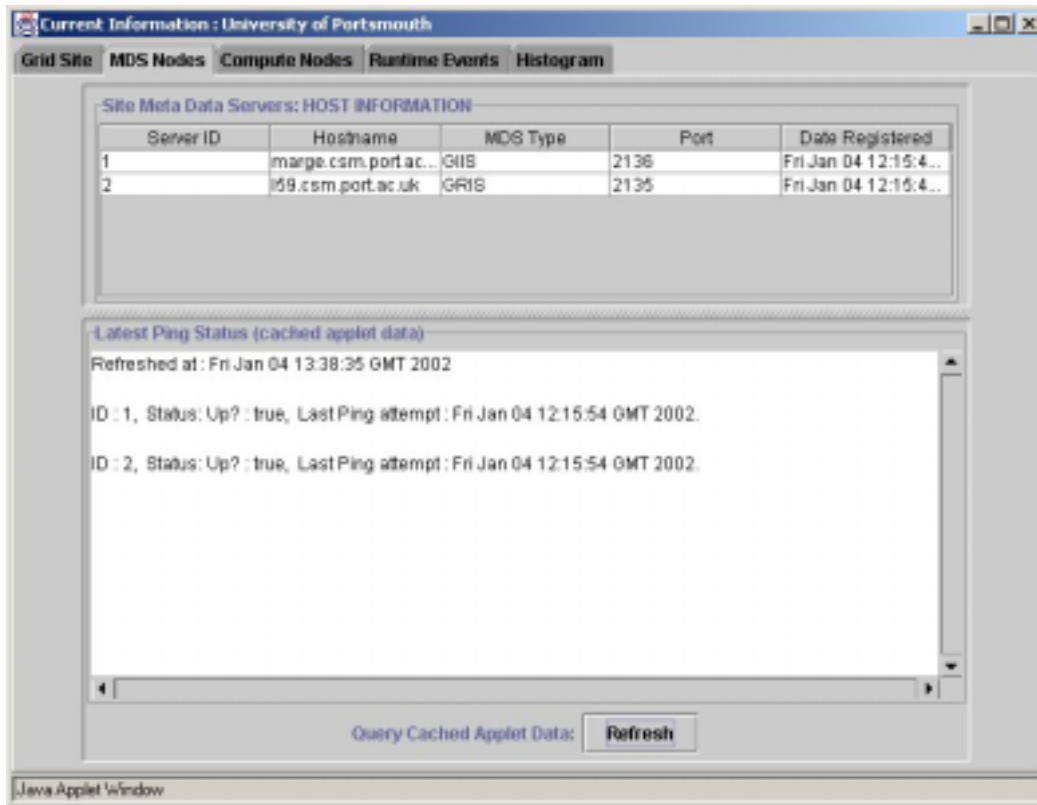


Figure 5: Polling Information From MDS Nodes

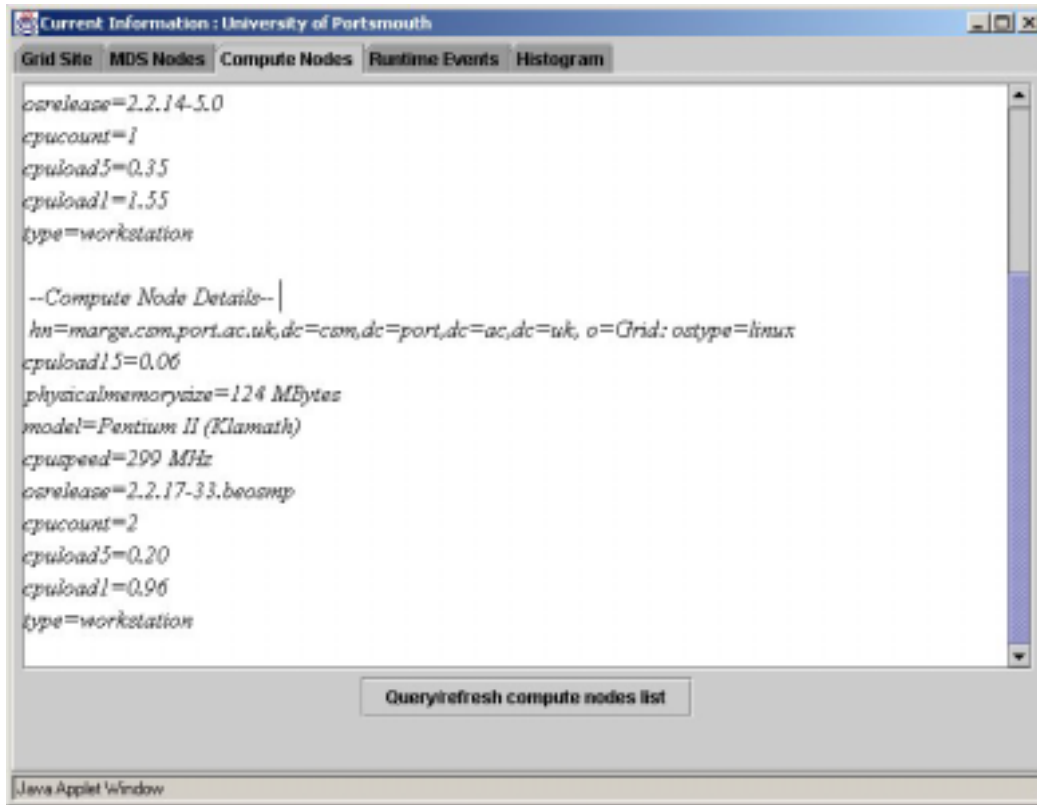


Figure 6: Query MDS for Compute Node Information

The data currently returned as compute node information (see Figure 6) is relatively basic in this version of the prototype. When the Query button is pressed the client sends a request to the servlet to execute an MDS query. The servlet attempts to fulfil the request and returns the result or error to the client. Currently MDS requests are executed on demand only.

This panel (in Figure 6) can be expanded in a later prototype to allow the monitoring and plotting of individual compute resources in greater detail. For example, graphs displaying current and recent performance statistics (updated in real-time) or the monitoring of a subset of user specified performance counters. This data might be stored in the local database for later retrieval (and display) or presented to the user and updated as new data arrives.

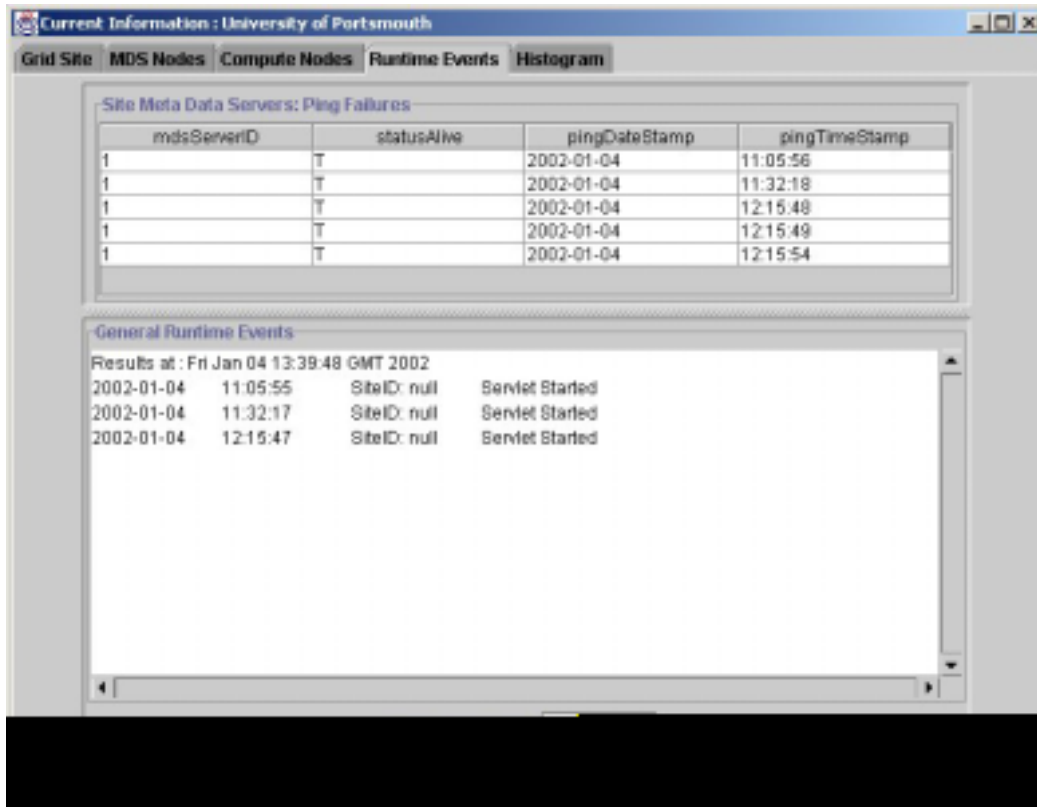


Figure 7: Runtime Events Displayed

Figure 7 presents two categories of information. The top part of the panel, displays information logged when the current site's MDS (or MDSs) failed to respond to a servlet probe attempt. For demonstration purposes, the applet has been configured to display those times when the primary MDS did respond. In addition, the polling frequency was adjusted at the servlet to a very high rate (for testing purposes, every 5 seconds!)

The lower part of the panel presents general runtime events. This information is primarily independent of the currently selected Grid site and provides a mechanism to observe the operating status of the servlet. For example, failure to connect with the local database might be displayed here and in the servlet log file.

## 5 Libraries and APIs

Name	Comment
Java2 [java02]	The Grid Monitor is entirely written in Java. All code is platform independent both at the server and the client.
Java Swing API [swing02]	Graphics API used for client graphical displays
Java Servlet API 1.0 [servlet02]	The servlet API is used as the framework for the server code: A HTTP servlet is implemented. Jakarta Tomcat [tomcat02] is used as the servlet engine.
MMmMySQL [mm02]	A Type IV JDBC driver (platform independent) for the MySQL database. LGPL License. Source code available.

Java CoG Kit 0.9.12 [cog02]	Provides access to Grid services through the Java framework. Used to retrieve data from Globus MDS
--------------------------------	--

**Table 3: Libraries and APIs**

## 6 Data Structures

The servlet and applet data structures are designed to provide a uniform framework. This approach is intended to minimise the amount of replicated data passed between servlet and client, thereby potentially improving system speed and scalability. Regardless of the amount of data passed to the client, access to this data is always consistent. Null object references within the data framework indicate missing data. The types of metadata used in the Grid Monitor are considered next. This is followed by a description of the implemented data structures.

### 6.1 Meta data

Each registered grid site contains four types of data:

1. Site registration data,
2. Static (Relatively) site data,
3. Dynamic site data,
4. Runtime data.

Site registration data is provided when a site's Grid administrator registers the site with the e-Science Support Centre. By its nature this information is static, although it may be updated infrequently. Therefore a mechanism is required to highlight alterations to clients so that any changes will be reflected within the client display.

Static (Relatively) and Dynamic site data, are obtained by querying each site's MDS. For example, static data might contain a cached list of machine names located in a site's MDS. This list of names (and associated platform details, i.e. OS type, CPU types, installed memory, etc) is unlikely to change often; therefore it is possible to reduce data transfer between client and servlet. This functionality has not currently been utilised within the applet or servlet code, but may be used in a later version. Dynamic runtime values associated with these machines are incorporated within separate dynamic data object containers. Dynamic data includes all data values that change during the normal operation of a Grid site. For example, performance counters: queue lengths, CPU load, or memory usage. Runtime data is maintained by the servlet. This data provides error information and timestamps of the time when a site was last successfully polled (or not!).

Site registration	Static	Dynamic	Runtime
Site ID (assigned during registration)	Node names	Node(s) processor load(s)	Timestamp of last successful MDS poll
Site Name	Node platform details	Node(s) memory usage	Servlet/site Communication Error messages
Address and Postcode	Node OS details	Node(s) Queue lengths	

GPS coordinates (optional)	Node job manager support	Node uptimes	
List of admin contact details		Availability of job managers on a node	
List of site MDS servers			
List of URLs to site related information			

**Table 4: Examples of Meta Data Grouping**

## 6.2 Data Structure Architecture

Because the same types of data are manipulated at both the servlet and applets, a common data structure architecture is used to promote uniformity and the reuse of a set of common classes. The common data structure is composed of many aggregate classes. These classes are used to group data by type, for example static registration data, dynamic poll data, etc. If partial data is returned to the client then references to unnecessary classes are initialised to null. This approach provides the opportunity for different types of information to be returned to the client in a uniform manner. Before utilising data from the returned container object, the client first checks for null references and reacts accordingly to the available data.

In order to utilise common data transport containers, additional attributes are used to identify the type and arguments for each type of data request. Servlet and clients then adjust their behaviour appropriately to deal with the incoming object.

## 7 Client-Server Communications

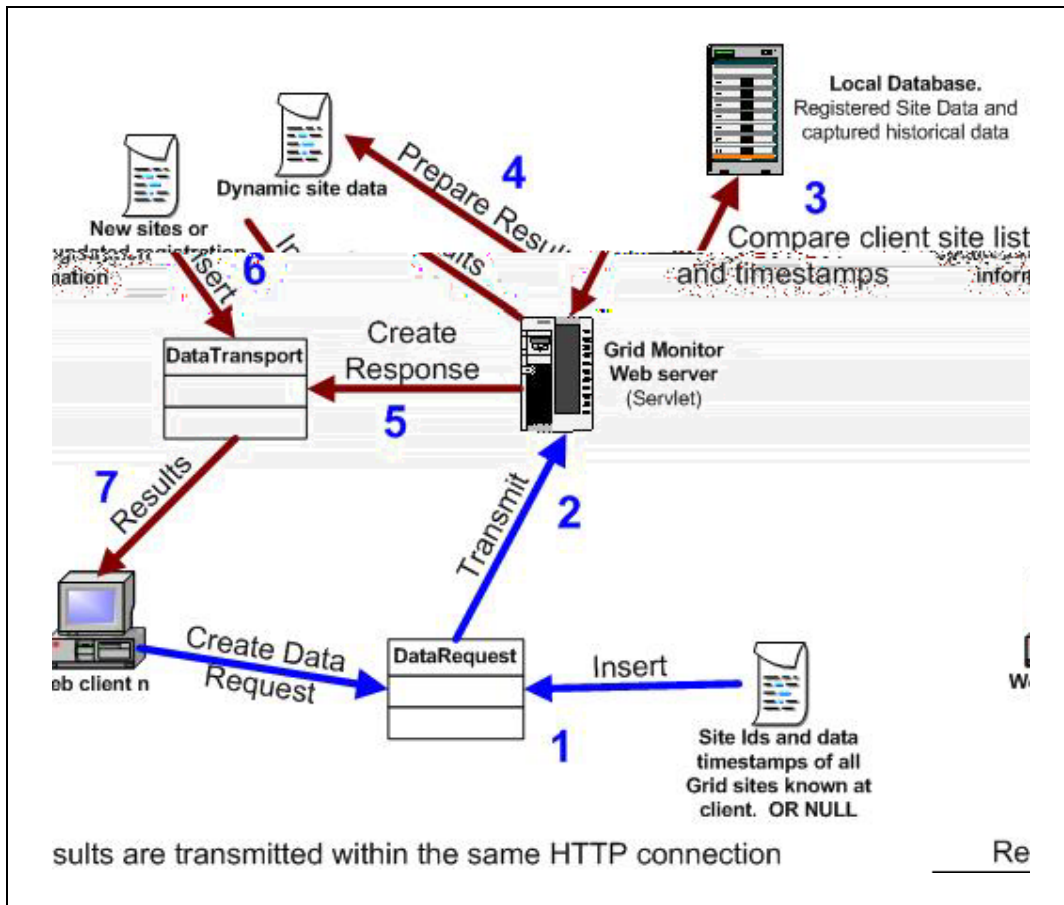


Figure 8: The Client Data Retrieval Mechanism

This section addresses the question of how clients determine which site data has been updated at the servlet and how these changes are propagated back to the client with the minimum amount of unnecessary data. Currently the prototype implements a subset of the following functionality (purely utilises empty client site lists within `DataRequest` objects). However, full support can be incorporated in future versions of the prototype.

Clients may have different frequencies at which they check the servlet for updates to site information. Furthermore, clients can vary these intervals during runtime to meet explicit requirements (such as reducing network traffic or CPU overhead at busy times). Each client sends a data request for dynamic (polled MDS) data to the servlet. The request contains a list of site IDs and timestamp pairs. This information indicates to the servlet the complete list of sites a client is aware of and the time that site information was last updated.

If a new site is registered with the servlet then the client list of site IDs will be incomplete. When returning site data, the servlet also returns any new sites (full data)

currently unknown at the client. On receipt, the client checks for new sites and appends them to the client data structures.

When receiving a client request for new data, the servlet must determine which sites have received new data. Because the data for each site is split into four types (each with a separate timestamp), checking four sets of timestamps for each site in a large list of sites will introduce additional overhead. In an attempt to reduce this overhead, a central timestamp is maintained in each site's data structure. This timestamp is updated whenever a change is made to any of the site's data. For example, if the performance data for a site is updated, then the performance data timestamps and the central site timestamp are updated as an atomic operation.

Each sites' data structure can be checked once to determine if a change has been made (instead of four times per site). If there has been no change to the timestamp then the next site's central timestamp can be inspected. If a site's timestamp has changed then it is necessary to inspect each of the four sub timestamps to determine which section has new information.

When a client initialises and connects to the servlet for the first time, a `DataRequest` object with an empty list of sites known by the applet is transmitted to the servlet. In response to an empty client site list, the servlet returns a `DataTransport` object containing full data (descriptive data and latest poll information) for all registered Grid sites. When the client next contacts servlet, the sites list (within the `DataRequest` object) contains entries for each of the sites the client is aware of. This metadata is lightweight, each site entry comprises a site ID and timestamps for when data for a given site entry was last updated.

On receiving this data, the servlet performs the following actions:

1. Use Site ID:
  - a. To locate all site entries,
  - b. To determine which new sites the client should know about,
2. Package up poll information for each site ready to be returned to the client.
3. Check if static data has changed, i.e. the number of MDS servers, names of administrators etc. Utilises timestamp comparisons.
4. All new information is returned to the client in a `DataTransport` object as necessary. If new data is not returned then object references within the data structure hierarchy are set to null appropriately.

Caching registered site data at the client and updating client data structures at a frequency that (at least) matches the rate polled data arrives at the servlet, presents a simple approach to propagating data alterations with a minimal network overhead.

## 7.1 Timestamps: Client-Server Data Control

In order to avoid issues with clock skew between client and server, the server's clock is used for all timestamp operations. On initialisation metadata (timestamps) are sent to the client indicating the last time data was updated by the servlet.

Subsequently, when clients contact the servlet to determine if new information must be downloaded, the previously stored timestamps are passed back to the servlet. This is then used by the servlet to construct a data object containing all new data. If a comparison of timestamps reveals that a block of data has not since been updated, then a null reference is inserted into the data object. When the client receives the data object, null references are ignored and only the new data inserted into the client data structures.

## 8 Servlet MDS Polling Approach

The servlet configuration file determines MDS polling frequency. Multiple sites will normally be registered with the Grid Monitor. In order to present consistent data updates from all sites, it is intended that the Grid Monitor will poll all sites concurrently using a thread for each Grid site – at the time of writing the MDS polling code is single threaded. It is intended that each thread will work through all MDS servers in sequence for a given site. Servlet data structures are updated as information arrives. The data structures are organised so that multiple threads can update data (for different Grid sites) concurrently, without interfering with updates for other sites.

It is recognised that interleaving MDS polling requests over time would reduce the impact of network and server performance. This approach may be modified in a later version of the Grid monitoring software if actual operating levels prove to be unsatisfactory.

### 8.1 MDS Polling Mechanism

Raw sockets are not supported under Java 2 (v1.3); it is also not possible to construct the ICMP datagram required to implement a pure Java version of the network utility Ping. Instead of introducing platform specific code (native methods), the `TcpPing` class has been utilised to determine whether a host is responding to network traffic. The `TcpPing` class is used to create a socket connection on a specified port. A successful socket connection implies a daemon is listening on the destination socket. The Grid Monitor attempts to connect to each MDS on the registered port. If a connection attempt fails, then the echo port can be probed to further determine the availability of the MDS. For example, this two-stage approach can determine:

1. Is the MDS daemon responding?
2. If the MDS daemon is not responding, is the host itself responding to other network traffic?
3. If the host is alive, then perhaps the MDS Daemon is dead, listening on a different port, or a packet filtering policy is in effect.

### 8.2 Polling Periods

The servlet configuration file is read before each scheduled MDS poll. Therefore, changes to the configuration file (for example the MDS poll frequency) can be used to dynamically adjust servlet behaviour at runtime. A servlet administration, Web interface can be implemented in a later prototype to permit easy access and configuration for the servlet runtime operation.

Currently, each Grid Monitoring client is constrained to use the same minimum frequency when polling the servlet for new information (i.e. Grid site MDS availability). This frequency is set at the time each individual client is initialised. Therefore, clients started at different times, although sharing the same poll frequency, will request new data from the servlet at different wall clock times. Clearly, the polling frequency of a client should not be less than the MDS polling frequency of the servlet; the client would potentially be polling for data before the servlet has been scheduled to capture new MDS data.

On an individual basis, each client is free to adjust the period between data requests to the servlet. Some clients may require five minute updates, while others may be satisfied with half hourly updates. Clearly, longer frequencies reduce servlet load and network traffic, but also reduce the timeliness of dynamic performance data retrieved from the servlet. If a client increases the latency between information updates, this affects that client only. The servlet will continue to poll the MDS data at the period defined in the servlet configuration file. Each client can adjust the rate of information updates independently.

## **9 Local Database**

All registration and servlet polled information is stored in the local Grid Monitor relational database. During normal operation, data structures within the servlet, cache a subset of the static and dynamic data in order to reduce database access overheads. In the event of a servlet crash, the latest data in the database can be read to help reduce the time required for the servlet to return to normal operation. Issues relating to data timeliness apply, but this approach may be useful when a long period between site polls is employed.

The database contains three table types, which hold primarily static, dynamic or runtime data. Data contained within tables of static data are expected to change infrequently. Data in these tables may be, for example, the names of resource job managers. Tables containing dynamic data can be expected to change after each MDS probe, as new values for performance counters are read in. Runtime data describes errors that occur when attempting to connect to an MDS for example. This table partitioning is intended reflect the way data is grouped (to minimise transport overhead) when data is passed from the servlet to clients.

### **9.1 Database Tables**

1. RegistrationTable,
2. AdminTable,
3. AdminContactTable,
4. MdsServerTable,
5. SiteUrlTable,
6. MdsStatusHistoryTable,
7. RuntimeEventsTable.

See Appendix A: Local Database Configuration, for details of table construction. Section 9.1.1.1 explains a number of points regarding the inclusion of some tables.

### 9.1.1.1 Database Table Explanations

The **AdminTable** is used to provide details of Grid administrators and key personal. A site may have multiple Grid administrators; equally multiple sites may share a single grid administrator.

Name	Data type	Description
adminID	Integer	Administrator's ID. Assigned during registration period. <b>Primary key</b>
Name	String	Administrator's name
position	String	Rank, Title or position held by administrator

**Table 5: AdminTable – Database Table**

The **AdminContactTable** is required to keep track of data for each site that a mobile administrator will work. For example, contact data may change as a result of individual site policies (for example a company mobile phone or pager issued by a given site). This table can also be queried to determine which administrators work at which sites (and which administrators work at multiple sites).

Name	Data type	Description
AdminId	Integer	Administrator's ID. Assigned during registration period. <b>Primary key</b>
SiteId	Integer	Which site are they currently located at? <b>Primary key</b>
EmailAddress	String	Email address for this site
TelephoneNumber	String	
MobileNumber	String	
FaxNumber	String	

**Table 6: AdminContactTable – Database Table**

**MdsServerTable** contains details of MDS servers. Each site can have multiple MDSs listed. These are arranged by Integer to indicate the relative importance of each server. A server with a low ID (such as 1) will be queried before a server with a higher number at the same site.

Name	Data type	Description
mdsServerId	Integer	(assigned at registration time) <b>Primary key</b> (i.e. 1 – 5. The primary server would be 1.)
SiteID	Integer	Primary key
serverHostname	String	Name of server
typeOfServer	String	GRIS or GIIS?
mdsPortNumber	Integer	Which port number to connect to?
dateRegistered	TimeStamp	When first registered with the Grid Monitor?

Table 7: MdsServerTable – Database Table

## 10 Future Work

It is intended that the initial prototype can be extended to provide greater functionality. For example:

- Limited configuration is currently available. This can be increased in a later version.
- Functionality can be extended to include dynamic compute resource performance graphing and network performance information between Grid sites.

Further modifications include:

### 10.1 Security

Currently no authentication is required to use the Grid Monitor: The Globus 1.1.3 and 1.1.4 MDS do not utilise authentication or encryption. However, security features have been introduced into newer versions of the Globus MDS. The Grid Monitor will need to operate with these secure MDS and therefore further investigation is needed. It is known that the latest version (currently beta) of Jakarta Tomcat supports GSI security [gsi02]; this may provide the mechanisms by which the Grid Monitor implements Grid security between the servlet and MDS servers. The current applet/servlet HTTP based communication model can be extended to utilise secure sockets.

### 10.2 Plotting Map Positions

Currently the coordinates of a Grid site on a map are determined manually in advance, usually when the site registration data is received. If the Grid Monitor is to be extended to use multiple levels of maps then the use of GPS data to automate this process for any map may be beneficial.

### 10.3 The User Interface

The client display requires updating to allow the plotting of compute resource performance indicators. The plotting functionality may allow pseudo real-time and historical data to be viewed. Compute resource data retrieved from the MDS may be formatted in a manner that allows user defined compute resource values to be monitored.

Control panels can be implemented within client displays to allow applet customisation during runtime (i.e. to change the frequency of the client's requests to the servlet).

### 10.4 Further Multi-Threading

The tabbed panes displaying site information are currently single threaded. Within the tabbed panes, a user can execute a long running task; for example, query all computer resources in a MDS. Until the task has completed, it is not possible to move to another tabbed pane in the *same window*.

The MDS site polling mechanism at the servlet requires minor modification. Currently all sites are polled from a single thread. The mechanism can be extended to allow concurrent polling of sites. A policy of one thread per site, or a thread pool may be selected, depending on the number of registered grid sites. The appropriate policy can be selected during servlet initialisation, based on the number of registered sites in the local database and servlet configuration rules.

## 11 Summary

This report presents a three-tier Grid monitoring prototype that displays Grid site location and MDS status over a graphical map. Site information, MDS availability and compute resource MDS entries are presented to the user. Polled data is recorded to a local database for historical analysis. The prototype is intended to gather Grid site information from multiple Grid sites into a single, easy to use utility.

The prototype was implemented in Java using applet, servlet, database technologies as well as the Globus Commodity Grid (CoG) APIs. The prototype code is platform independent as it is written with Java-based technologies. Java applets were used to reduce server load, by moving responsibility for GUI processing to the clients. A multi-level data structure was implemented to allow the optimisation of data communications between clients and the servlet. Caching and servlet timestamp comparisons were used to reduce network traffic by removing unnecessary data from returning client data requests. In order to permit client access through firewalls, a client-pull approach utilising HTTP serialised objects is used for communicating with the servlet.

Further work on the prototype is required to implement the automated monitoring (and visualisation) of compute resources within a Grid site MDS. Network monitoring between Grid sites could equally be included in a later version.

## 12 References

- [uk-grid-support01] GSC- UK Grid Support Centre, <http://www.grid-support.ac.uk>, January 2001.
- [e-science02] The e-Science Core Programme, <http://www.escience-grid.org.uk/>, January 2002.
- [globus-mds02] Metadatacomputing Directory Service, The Globus Project, <http://www.globus.org/mds/>, January 2002.
- [mysql02] MySQL, <http://www.mysql.org/>, January 2002.
- [cog02] Globus Java CoG Kit, The Globus Project, <http://www.globus.org/cog/java/>, January 2002.
- [java02] Java2 SDK, Standard Edition Product Family, <http://java.sun.com/j2se/>, January 2002.
- [gsi02] Globus Security Infrastructure, The Globus Project,

[servlet02]	<a href="http://www.globus.org/security/">http://www.globus.org/security/</a> , January 2002.
[tomcat02]	Java Servlet Technology, <a href="http://java.sun.com/products/servlet/">http://java.sun.com/products/servlet/</a> , January 2002.
[globus02]	Jakarta Project, Jakarta Tomcat 3.3, <a href="http://jakarta.apache.org/tomcat/">http://jakarta.apache.org/tomcat/</a> , January 2002.
[mm02]	The Globus Project, <a href="http://www.globus.org/">http://www.globus.org/</a> , January 2002.
[swing02]	MM MySQK JDBC Driver, <a href="http://mmmmysql.sourceforge.net/">http://mmmmysql.sourceforge.net/</a> , January 2002.
[ppepc01]	Java Swing API, <a href="http://java.sun.com/products/">http://java.sun.com/products/</a> , January 2002.
	University of Glasgow, GridPP project, Grid Monitor, <a href="http://ppepc37.ph.gla.ac.uk:8001/">http://ppepc37.ph.gla.ac.uk:8001/</a> , January 2002.

## 13 Appendix A: Local Database Configuration

Selected database: MySQL. MySQL does not support the SQL foreign key concept or concatenated primary keys. This is reflected in the syntax below where concatenated unique indexes are used to simulate compound primary keys.

### 13.1 SQL Syntax

#### Create the database

1. Create the database: `mysqladmin -u root -p create GRID_MONITOR`

#### Create the tables

1. `mysql -u root -p`
2. `use GRID_MONITOR`

```
mysql> CREATE TABLE RegistrationTable ( siteID INT NOT NULL
PRIMARY KEY,
-> siteName VARCHAR(100),
-> siteAddress1 VARCHAR(100),
-> siteAddress2 VARCHAR(100),
-> siteAddress3 VARCHAR(100),
-> siteAddress4 VARCHAR(100),
-> siteTown VARCHAR(50),
-> siteCounty VARCHAR(30),
-> sitePostalCode VARCHAR(15),
-> siteCountry VARCHAR(30),
-> gpsCoordinates VARCHAR(20),
-> gpsScheme VARCHAR(20),
-> defaultHelpDeskEmail VARCHAR(30),
-> defaultHelpDeskPhone VARCHAR(30))
-> ;
```

This table auto increments with each new entry: Do not supply AdminIDs for this table.

```
mysql> CREATE TABLE AdminTable ( adminID INT NOT NULL
PRIMARY KEY AUTO_INCREMENT,
```

```
-> firstName VARCHAR(50),
-> surname VARCHAR(50),
-> position VARCHAR(30));
```

```
mysql> CREATE TABLE AdminContactTable( adminID INT NOT NULL,
-> siteID INT NOT NULL,
-> emailAddress VARCHAR(30),
-> telephoneNumber VARCHAR(30),
-> mobileNumber VARCHAR(30),
-> faxNumber VARCHAR(30),
-> UNIQUE key (adminID, siteID));
```

```
mysql> CREATE TABLE MdsServerTable( mdsServerID INT NOT NULL,
-> siteID INT NOT NULL,
-> serverHostname VARCHAR(50),
-> typeOfServer VARCHAR(20),
-> mdsPortNumber INT,
-> dateRegistered DATETIME,
-> UNIQUE key(mdsServerID,siteID));
```

```
mysql> CREATE TABLE SiteUrlTable( siteID INT NOT NULL,
-> urlID INT NOT NULL,
-> title VARCHAR(50),
-> url VARCHAR(200),
-> UNIQUE key(siteID,urlID));
```

```
mysql> CREATE TABLE MdsStatusHistoryTable( mdsServerID INT NOT NULL,
-> siteID INT NOT NULL,
-> statusAlive CHAR(1) NOT NULL,
-> pingDateStamp DATE NOT NULL,
-> pingTimeStamp TIME NOT NULL,
-> UNIQUE key (mdsServerID, siteID, pingDateStamp, pingTimeStamp));
```

```
mysql> CREATE TABLE RuntimeEventsTable(id INT NOT NULL PRIMARY
KEY AUTO_INCREMENT,
-> eventDateStamp DATE NOT NULL,
-> eventTimeStamp TIME NOT NULL,
-> eventTypeID INT NOT NULL,
-> siteID INT,
-> mdsID INT,
-> eventMessage TINYTEXT);
```

## 13.2 Additional Table Definitions

Aspects of the following tables may be used when implementing dynamic monitoring (and logging) of MDS compute resources.

Name	Data type	Description
------	-----------	-------------

nodeId	String	Primary key Qualified host name Could use mac addresses for this, but network cards get changed over and replaced!!
hostname	String	
objectname	String	
manufacturer	String	
Model	String	
machineHardwareName	String	
type	String	i.e. Workstation
osrelease	String	OS release version information
ostype	String	
physicalInstalledMemory	String	Amount of memory installed
cpuType	String	CPU details
cpuCount	Int	
cpuSpeed	String	
contactString	String	The full contact name for the jobmanager (including port)
gramVersion	String	
gramVersionDate	String	
gramSecurity	String	
lastGramUpdate	TimeStamp	
gramTtl	time	
deployDir	String	Globus deploy directory

### JobmanagerTable

Name	Data type	Description
nodeId	String	Primary key Qualified host name Could use MAC addresses for this, but network cards get changed over and replaced!!
schedulerType	String	Primary key
description	String	Description
schedulerVersion	String	

### 13.2.1 Dynamic Tables

**GeneralNodePerformanceTable** contains data reflecting the general utilisation of a node at a given time.

Name	Data type	Description
timestamp		Primary key
nodeId		Primary key
processorLoad-1	Integer	<b>CPU load for past 1 minute</b>

processorLoad-5	Integer	<b>CPU load for past 5 minutes</b>
processorLoad-15	Integer	<b>CPU load for past 15 minutes</b>

**QueuePerformanceTable** stores status and performance data.

Name	Data type	Description
timestamp	Timestamp	Primary key
nodeId ( <i>ComputeNodesTable.nodeId</i> )	Integer	Primary key ( <b>primary key also in ComputeNodesTable</b> )
queueName	String	Primary key
maxtime	Integer	
maxCpuTime		
maxCount		
maxRunningJobs		
maxJobsInQueue		
maxTotalMemory		
maxSingleMemory		
totalNodes		
freeNodes		
whenActive		
status		
dispatchType		
priority		
jobWait		
schedulerSpecific		
TTL		
lastUpdate		